Question 2dii: Explain your answer to the previous part (Question 2di) based on your knowledge from lectures and details from the query plans. Your explanation should also include why you didn't choose certain options. Please answer in maximum 5 sentences.

Because views and materialized views help us utilize pre-run queries rather than query from the base tables, they will reduce both cost and execution time compared to without views (option A and D). Additionally, materialized views are stored on disk and do not need to run again whenever it is called unline views (run on demand), materialized views take less time to create than a view.

0.1 Question 3c:

Given your findings from inspecting the query plans of queries from Questions 3a and 3b, fill in the blank and **justify your answer**. Explain your answer based on your knowledge from lectures, and details from the query plans (your explanation should include why you didn't choose other options). Your response should be no longer than 3 sentences.

Note: Your answer should be formatted as follows: `A because ...`

Adding a filter _____ the cost. A. increased B. decreased C. did not change

B because adding a filter will remove more rows from a relation; hence, lower cost when scanning the relation.

0.2 Question 3d:

Given your findings from inspecting the query plans of queries from Questions 3a and 3b, fill in the blank and **justify your answer**. Explain your answer based on your knowledge from lectures, and details from the query plans (your explanation should include why you didn't choose other options). Your response should be no longer than 3 sentences.

Note: Your answer should be formatted as follows: `A because ...`

Adding a filter _____ the execution time. A. increased B. decreased C. did not change

B because adding a filter will remove more rows from a relation; similarly to 3c, lower execution time when scanning the relation.

0.3 Question 4d

Given your findings above, why did the query optimizer ultimately choose the specific join approach you found in each of the above three scenarios in Questions 4a, 4b, and 4c? Feel free to discuss the pros and cons of each join approach as well.

If you feel stuck, here are some things to consider: Does a non-equijoin constrain us to certain join approaches? What's an added benefit in regards to the output of merge join?

Note: Your answer should be formatted as follows: `Q4a: A because ... Q4b: A because ...` You should write no more than 5 sentences.

Q4a: C Hash Join because we are doing equijoin on two large tables. Q4b: B Merge-Join because we want our output to be sorted, so the merged-join already sort the relation for us while doing the join, we don't have to do additional sort. Q4c: A Nested Loop Join because the non-equijoin constrains us from using hash join, so the nested-loop is the optimal solution here.

0.3.1 Question 5di Justification

Explain your answer to `Question 5d` above based on your knowledge from lectures, and details from inspecting the query plans (your explanation should include why you didn't choose certain options). Your answer should be no longer than 3 sentences.

When we index on g_batting, it will help the filter during joining 2 tables faster by pushing down predicates (g_batting = 10) before joining; hence having significant impact on both query cost and execution time. On the other hand, salary index has no impact on the joining process, so the cost and query time stay the same or has negligible changes.

0.3.2 Question 6ei Justification

Explain your answer to `Question 6e` above based on your knowledge from lectures, and details from inspecting the query plans (your explanation should include why you didn't choose certain options). Your answer should be no longer than 3 sentences.

When adding an index on an AND predicate, one of the predicate got filtered will also reduce the number of tuples for the second filter; hence, it reduce both query cost and execution time. On the other hand, in OR predicate, adding an index will only impact on one filter while the other filter will essentially scan all the relation; hence the cost and time stay the same. Lastly, adding a multicolumn index on columns that got filtered on both predicates of OR statement, so both of them will effeciently got filtered. Thus, reducing both query cost and execution time.

0.4 Question 7c

Given your findings from `Question 7`, which of the following statements is true? A. An index on the column being aggregated in a query will always provide a performance enhancement. B. A query finding the MIN(salary) will always benefit from an index on salary, but a query finding MAX(salary) will not. C. A query finding the COUNT(salary) will always benefit from an index on salary, but a query finding AVG(salary) will not. D. Queries finding the MIN(salary) or MAX(salary) will always benefit from an index on salary, but queries finding AVG(salary) or COUNT(salary) will not.

Justify your answer. Explain your answer based on your knowledge from lectures, and details of the query plans (your explanation should include why you didn't choose certain options). Your response should be no longer than 3 sentences.

Note: Your answer should be formatted as follows: "A because …"

D because MIN(salary) and MAX(salary) will look up to a specific value, hence benifiting from index while AVG(salary) and COUNT(salary) still scan through all the relation, so the index will not help in this situation (resulting in sequential scan despite having an index).

0.5 Question 9c:

What difference did you notice when you added an index into the salaries table and re-timed the update? Why do you think it happened? Your answer should be no longer than 3 sentences.

When adding an index into the salaries table, the query time increases because the new information has to be written into both relations and indexes.

1 Question 10: Project Takeaways

In this project, we explored how the database system optimizes query execution and how users can futher tune the performance of their queries.

Familiarizing yourself with these optimization and tuning methods will make you a better data engineer. In this question, we'll ask you to recall and summarize these concepts. Who knows? Maybe one day it will help you during an interview or on a project.

In the following answer cell, 1. Name 3 methods you learned in this project. The method can be either the optimization done by the database system, or the fine tuning done by the user. 2. For each method, summarize how and why it can optimize query performance. Feel free to discuss any drawbacks, if applicable.

Your answer should be no longer than ten sentences. Each method identification/discussion is 2 points.

1. Adding indexes helps with filtering and MIN(), MAX() operators. Index helps the database engine find the data faster, so when performing filtering on one single exact data, in AND statements, it will push down the predicates and reduce the number of operations done by removing irrelavent rows. Similarly, index can be befinicial when finding min or max on the indexed column, but not COUNT() or AVG(). Additionally, indexing on one column does not affect OR statements, so we need to consider multipleindex. This also comes with the cost of index management that it will be expensive when more data comes in later on.
2. Clustered index will help with filtering on a range of data. Unlike normal index, clusered index will rearrange the affected column, so filtering on a large range of data will hugely benefit from this ordered database. Cost of management also has to be considered.
3. When it comes to joining tables, hash joins and merge-joins perform well on equi-joining tables, and nested loops are utilized on non-equijoining tables. Also, nested-loops are optimal when one of the relation is much smaller. Moreover, merge-joins are mostly used when the output needs to be sorted, and hash-joins are suitable for large database.